GameFormer: Game-theoretic Modeling and Learning of Transformer-based Interactive Prediction and Planning for Autonomous Driving

Supplementary Material

A. Experiment Details

A.1. Prediction-oriented Model

Model inputs. In each scene, one of the two interacting agents is designated as the focal agent, with its current state serving as the origin of the coordinate system. We consider 10 surrounding agents closest to a target agent as the background agents, and therefore, there are two target agents to predict and up to 20 different background agents in a scene. The current and historical states of each agent are retrieved for the last one second at a sampling rate of 10Hz, resulting in a tensor with a shape of (22×11) for each agent. The state at each timestep includes the agent's position (x, y), heading angle (θ), velocity (v_x, v_y), bounding box size (L, W, H), and one-hot category encoding of the agent (totally three types). All historical states for each agent are aggregated into a fixed-shape tensor of $(22 \times 11 \times 11)$, with missing agent states padded as zeros, to form the input tensor of historical agent states.

For each target agent, up to 6 drivable lanes (each extending 100 meters) that the agent may take are identified using depth-first search on the road graph, along with 4 nearby crosswalks as the local map context, with each map vector containing 100 waypoints. The features of a waypoint in a drivable lane include the position and heading angles of the centerline, left boundary, and right boundary, speed limit, as well as discrete attributes such as the lane type, traffic light state, and controlled by a stop sign. The features of a waypoint in the crosswalk polyline only encompass position and heading angle. Therefore, the local map context for a target agent comprises two tensors: drivable lanes with shape $(6 \times 100 \times 15)$ and crosswalks with shape $(4 \times 100 \times 3)$.

Encoder structure. In the encoder part, we utilize two separate LSTMs to encode the historical states of the target and background agents, respectively, resulting in a tensor with shape (22×256) that encompasses all agents' historical state sequences. The local map context encoder consists of a lane encoder for processing the drivable lanes and a crosswalk encoder for the crosswalk polylines. The lane encoder employs MLPs to encode numeric features and embedding layers to encode discrete features, outputting a tensor of encoded lane vectors with shape $(2 \times 6 \times 100 \times 256)$, while the crosswalk encoder uses an MLP to encode numeric features, resulting in a tensor of crosswalk vectors with shape $(2 \times 4 \times 100 \times 256)$. Subsequently, we utilize a max-pooling layer (with a step size of 10) to aggregate the waypoints from a drivable lane in the encoded lane tensor, yielding a tensor with shape $(2 \times 6 \times 10 \times 256)$ that is reshaped to $(2 \times 60 \times 256)$. Similarly, the encoded crosswalk tensor is processed using a max-pooling layer with a step size of 20 to obtain a tensor with shape $(2 \times 20 \times 256)$. These two tensors are concatenated to produce an encoded local map context tensor with shape $(2 \times 80 \times 256)$. For each target agent, we concatenate its local map context tensor with the historical state tensor of all agents to obtain a scene context tensor with dimensions of (102×256), and we use self-attention Transformer encoder layers to extract the relationships among the elements in the scene. It is important to note that invalid positions in the scene context tensor are masked from attention calculations.

Decoder structure. For the $M = 6$ joint prediction model, we employ the learnable latent modality embedding with a shape of $(2 \times 6 \times 256)$. For each agent, the query (6×256) in the level-0 decoder is obtained by summing up the encoding of the target agent's history and its corresponding latent modality embedding; the value and key are derived from the scene context by the encoder. The level-0 decoder generates the multi-modal future trajectories of the target agent with x and y coordinates using an MLP from the attention output. The scores of each trajectory are decoded by another MLP with a shape of (6×1). In a level-k decoder, we use a shared future encoder across different layers, which includes an MLP and a max-pooling layer, to encode the future trajectories from the previous level into a tensor with a shape of (6×256). Next, we employ the trajectory scores to average-pool the encoded trajectories, which results in the encoded future of the agent. The encoded futures of the two target agents are then fed into a self-attention Transformer layer to model their future interaction. Finally, the output of the Transformer layer is appended to the scene context obtained from the encoder.

For the $M = 64$ marginal prediction model, we use a set of 64 fixed intention points that are encoded with MLPs to create the modality embedding with shape $(2 \times 64 \times 256)$. This modality embedding serves as the query input for the level-0 decoder. The fixed intention points are obtained through the K-means method from the training dataset. For each scene, the intention points for the two target agents are normalized based on the focal agent's coordinate system. The other components of the decoder are identical to those used in the joint prediction model.

Training. In the training dataset, each scene contains several agent tracks to predict, and we consider each track sequentially as the focal agent, while the closest track to

the focal agent is chosen as the interacting agent. The task is to predict six possible joint future trajectories of these two agents. We employ only imitation loss at each level to improve the prediction accuracy and training efficiency.

In the joint prediction model, we aim to predict the joint and scene-level future trajectories of the two agents. Therefore, we backpropagate the loss through the joint future trajectories of the two agents that most closely match the ground truth (i.e., have the least sum of displacement errors). In the marginal prediction model, we backpropagate the imitation loss to the individual agent through the positive GMM component that corresponds to the closest intention point to the endpoint of the ground-truth trajectory.

Our models are trained for 30 epochs using the AdamW optimizer with a weight decay of 0.01. The learning rate starts with 1e-4 and decays by a factor of 0.5 every 3 epochs after 15 epochs. We also clip the gradient norm of the network parameters with the max norm of the gradients as 5. We train the models using 4 NVIDIA Tesla V100 GPUs, with a batch size of 64 per GPU.

Testing. The testing dataset has three types of agents: vehicle, pedestrian, and cyclist. For the vehicle-vehicle interaction, we randomly select one of the two vehicles as the focal agent. For other types of interaction pairs (*e.g.*, cyclist-vehicle and pedestrian-vehicle), we consider the cyclist or pedestrian as the focal agent. For the marginal prediction model, we employ the Expectation-Maximization (EM) method to aggregate trajectories for each agent. Specifically, we use the EM method to obtain 6 marginal trajectories (along with their probabilities) from the 64 trajectories predicted for each agent. Then, we consider the top 6 joint predictions from the 36 possible combinations of the two agents, where the confidence of each combination is the product of the marginal probabilities.

A.2. Planning-oriented Model

Model inputs. In each scene, we consider the AV and 10 surrounding agents to perform planning for the AV and prediction for other agents. The AV's current state is the origin of the local coordinate system. The historical states of all agents in the past two seconds are extracted; for each agent, we find its nearby 6 drivable lanes and 4 crosswalks. Additionally, we extract the AV's traversed lane waypoints from its ground truth future trajectory and use a cubic spline to interpolate these waypoints to generate the AV's reference route. The reference route extends 100 meters ahead of the AV and contains 1000 waypoints with 0.1 meters intervals. It is represented as a tensor with shape (1000×5). The reference route tensor also contains information on the speed limit and stop points in addition to positions and headings.

Model structure. For each agent, its scene context tensor is created as a concatenation of all agents' historical states and encoded local map elements, resulting in a tensor of shape (91×256). In the decoding stage, a learnable modality embedding of size $(11 \times 6 \times 256)$ and the agent's historical encoding are used as input to the level-0 decoder, which outputs six possible trajectories along with corresponding scores. In the level-k decoder, the future encodings of all agents are obtained through a self-attention module of size (11×256), and are concatenated with the scene context tensor from the encoder. This concatenation generates an updated scene context tensor with a shape of (102×256). When decoding an agent's future trajectory at the current level, the future encoding of that agent in the scene context tensor is masked to avoid using its previously predicted future information.

Training. In data processing, we filter those scenes where the AV's moving distance is less than 5 meters (*e.g.*, when stopping at a red light). Similarly, we perform joint future prediction and calculate the imitation loss through the joint future that is closest to the ground truth. The weights for the imitation loss and interaction loss are set to $w_1 = 1, w_2 = 0.1$. Our model is trained for 20 epochs using the AdamW optimizer with a weight decay of 0.01. The learning rate is initialized to 1e-4 and decreases by a factor of 0.5 every 2 epochs after the 10th epoch. We train the model using an NVIDIA RTX 3080 GPU, with a batch size of 32.

Testing. The testing scenarios are extracted from the WOMD, wherein the ego agent shows dynamic driving behaviors[1]. In open-loop testing, we check collisions between the AV's planned trajectory and other agents' ground-truth future trajectories, and we count a miss if the distance between AV's planned state at the final step and the ground-truth state is larger than 4.5 meters. The planning errors and prediction errors are calculated according to the most-likely trajectories scored by the model. In closed-loop testing, the AV plans a trajectory at every timestep with an interval of 0.1 seconds and executes the first step of the plan.

A.3. Baseline Methods

To compare model performance, we introduce the following learning-based planning baselines.

Vanilla Imitation Learning (IL): A simplified version of our model that directly outputs the planned trajectory of the AV without explicitly reasoning other agents' future trajectories. The plan is only a single-modal trajectory. The original encoder part of our model is utilized, but only one decoder layer with the ego agent's historical encoding as the query is used to decode the AV's plan.

Deep Imitative Model (DIM): A probabilistic planning method that aims to generate expert-like future trajectories $q(\mathbf{S}_{1:T}|\phi) = \prod_{t=1}^{T} q(\mathbf{S}_t|\mathbf{S}_{1:t-1}, \phi)$ given the AV's obser-

[1] https://github.com/smarts-project/smarts-project.offline-datasets/blob/master/waymo_candid_list.csv

vations ϕ. We follow the original open-source DIM implementation and use the rasterized scene image $\mathbb{R}^{200\times200\times3}$ and the AV's historical states $\mathbb{R}^{11\times5}$ as the observation. We use a CNN to encode the scene image and an RNN to encode the agent's historical states. The AV's future state is decoded (as a multivariate Gaussian distribution) in an autoregressive manner. In testing, DIM requires a specific goal $\mathcal{G}$ to direct the agent to the goal, and a gradient-based planner maximizes the learned imitation prior $\log q\left(\mathbf{S}|\phi\right)$ and the test-time goal likelihood $\log p(\mathcal{G}|\mathbf{S},\phi)$.

Robust Imitative Planning (RIP): An epistemic uncertainty-aware planning method that is developed upon DIM and shows good performance in conducting robust planning in out-of-distribution (OOD) scenarios. Specifically, we employ the original open-source implementation and choose the worst-case model that has the worst likelihood $\min_d \log q\left(\mathbf{S}_{1:T}|\phi\right)$ among $d=6$ trained DIM models and improve it with a gradient-based planner.

Conservative Q-Learning (CQL): A widely-used offline reinforcement learning algorithm that learns to make decisions from offline datasets. We implement the CQL method with the d3rlpy offline RL library[2]. The RL agent takes the same state inputs as the DIM method and outputs the target pose of the next step $(\Delta x, \Delta y, \Delta\theta)$ relative to the agent's current position. The reward function is the distance traveled per step plus an extra reward for reaching the goal, *i.e.*, $r_t = \Delta d_t + 10\times\mathbb{1}\left(d(s_t, s_{goal})<1\right)$. Since the dataset only contains perfect driving data, no collision penalty is included in the reward function.

Differentiable Integrated Prediction and Planning (DIPP): A joint prediction and planning method that uses a differentiable motion planner to optimize the trajectory according to the prediction result. We adopt the original open-source implementation and the same state input setting. We increase the historical horizon to 20 and the number of prediction modalities from 3 to 6. In open-loop testing, we utilize the results from the DIPP prediction network without trajectory planning (refinement).

MultiPath++: A high-performing motion prediction model that is based on the context-aware fusion of heterogeneous scene elements and learnable latent anchor embeddings. We utilize the open-source implementation of MultiPath++[3] that achieved state-of-the-art prediction accuracy on the WOMD motion prediction benchmark. We train the model to predict 6 possible trajectories and corresponding scores for the ego agent using the same dataset. In open-loop testing, only the most-likely trajectory will be used as the plan for the AV.

Motion Transformer (MTR)-e2e: A state-of-the-art prediction model that occupies the first place on the WOMD motion prediction leaderboard. We follow the original open-source implementation of the context encoder and MTR decoder. However, we modified the decoder to use an end-to-end variant of MTR that is better suited for the open-loop planning task. Specifically, only 6 learnable motion query pairs are used to decode 6 possible trajectories and scores. The same dataset is used to train the MTR-e2e model, and the data is processed according to the MTR context inputs.

[2] https://github.com/takuseno/d3rlpy

[3] https://github.com/stepankonev/waymo-motion-prediction-challenge-2022-multipath-plus-plus

A.4. Refinement Planner

Inverse dynamic model. To convert the initial planned trajectory to control actions $\{a_t, \delta_t\}$ (*i.e.*, acceleration and yaw rate), we utilize the following inverse dynamic model.

$$\begin{aligned}\Phi^{-1}: v_t &= \frac{\Delta p}{\Delta t} = \frac{\parallel p_{t+1} - p_t \parallel}{\Delta t},\\ \theta_t &= \arctan\frac{\Delta p_y}{\Delta p_x},\\ a_t &= \frac{v_{t+1}-v_t}{\Delta t},\\ \delta_t &= \frac{\theta_{t+1}-\theta_t}{\Delta t},\end{aligned} \tag{S1}$$

where p_t is a predicted coordinate in the trajectory, and Δt is the time interval.

Dynamic model. To derive the coordinate and heading $\{p_{x_t}, p_{y_t}, \theta\}$ from control actions, we adopt the following differentiable dynamic model.

$$\begin{aligned}\Phi: v_{t+1} &= a_t\Delta t + v_t,\\ \theta_{t+1} &= \delta_t\Delta t + \theta_t,\\ p_{x_{t+1}} &= v_t\cos\theta_t\Delta t + p_{x_t},\\ p_{y_{t+1}} &= v_t\sin\theta_t\Delta t + p_{y_t}.\end{aligned} \tag{S2}$$

Motion planner. We use a differentiable motion planner proposed in DIPP to plan the trajectory for the AV. The planner takes as input the initial control action sequence derived from the planned trajectory given by our model. We formulate each planning cost term c_i as a squared vector-valued residual, and the motion planner aims to solve the following nonlinear least squares problem:

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \frac{1}{2}\sum_i \parallel \omega_i c_i(\mathbf{u}) \parallel^2, \tag{S3}$$

where $\mathbf{u}$ is the sequence of control actions, and ω_i is the weight for cost c_i.

We consider a variety of cost terms as proposed in DIPP, including travel speed, control effort (acceleration and yaw rate), ride comfort (jerk and change of yaw rate), distance to the reference line, heading difference, as well as the cost of violating traffic light. Most importantly, the safety cost takes all other agents' predicted states into consideration and avoids collision with them, as illustrated in DIPP.

We use the Gauss-Newton method to solve the optimization problem. The maximum number of iterations is 30, and the step size is 0.3. We use the best solution during the iteration process as the final plan to execute.

Learning cost function weights. Since the motion planner is differentiable, we can learn the weights of the cost terms by differentiating through the optimizer. We use the imitation learning loss below (average displacement error and final displacement error) to learn the cost weights, as well as minimize the sum of cost values. We set the maximum number of iterations to 3 and the step size to 0.5 in the motion planner. We use the Adam optimizer with a learning rate of 5e-4 to train the cost function weights; the batch size is 32 and the total number of training steps is 10,000.

$$\mathcal{L} = \lambda_1 \sum_t ||\hat{s}_t - s_t||^2 + \lambda_2 ||\hat{s}_T - s_T||^2 + \lambda_3 \sum_i ||c_i||^2, \tag{S4}$$

where $\lambda_1 = 1, \lambda_2 = 0.5, \lambda_3 = 0.001$ are the weights.

A.5. GameFormer Planner

To validate our model's performance on the nuPlan benchmark[4], we have developed a comprehensive planning framework to handle the realistic driving scenarios in nuPlan. The planning process comprises the following steps: 1) feature processing: relevant data from the observation buffer and map API undergoes preprocessing to extract input features for the prediction model; 2) path planning: candidate route paths for the ego vehicle are computed, from which the optimal path is selected as the reference path; 3) model query: the prediction model is queried to generate an initial plan for the ego vehicle and predict the trajectories of surrounding agents; and 4) trajectory refinement: a nonlinear optimizer is employed to refine the ego vehicle's trajectory on the reference path and produce the final plan. For computational efficiency, we use a compact version of the GameFormer model, configuring it with 3 encoding layers and 3 decoding layers (1 initial decoding layer and 2 interaction decoding layers). Additionally, we introduce an extra decoding layer after the last interaction decoding layer to separately generate the ego vehicle's plan. The ego plan is then projected onto the reference path as an initialization of the refinement planner. The output of the GameFormer model consists of multimodal trajectories for the surrounding agents. For each neighboring agent, we select the trajectory with the highest probability and project it onto the reference path using the Frenet transformation, subsequently calculating spatiotemporal path occupancy. A comprehensive description of the planning framework can be found in this dedicated report[5].

[4] https://eval.ai/web/challenges/challenge-page/1856/overview

[5] https://opendrivelab.com/e2ead/AD23Challenge/Track_4_AID.pdf

B. Additional Quantitative Results

B.1. Interaction Prediction

Table S1 displays the per-category performance of our models on the WOMD interaction prediction benchmark, in comparison with the MTR model. The GameFormer joint prediction model exhibits the lowest minFDE across all object categories, indicating the advantages of our model and joint training of interaction patterns. Our GameFormer model surpasses MTR in the cyclist category and achieves comparable performance to MTR in other categories, though with a much simpler structure than MTR.

Table S1. Per-class performance of interaction prediction on the WOMD interaction prediction benchmark

Class	Model	minADE (↓)	minFDE (↓)	Miss rate (↓)	mAP (↑)
Vehicle	MTR	**0.9793**	2.2157	0.3833	**0.2977**
	GF (J)	0.9822	**2.0745**	**0.3785**	0.1856
	GF (M)	1.0499	2.4044	0.4321	0.2469
Pedestrian	MTR	**0.7098**	1.5835	**0.3973**	**0.2033**
	GF (J)	0.7279	**1.4894**	0.4272	0.1505
	GF (M)	0.7978	1.8195	0.4713	0.1962
Cyclist	MTR	1.0652	2.3908	**0.5428**	0.1102
	GF (J)	**1.0383**	**2.2480**	0.5536	0.0768
	GF (M)	1.0686	2.4199	0.5765	**0.1338**

B.2. nuPlan Benchmark

Table S2 presents a performance comparison between our planner and the DIPP planner. For the benchmark evaluation, we replace the prediction model in the proposed planning framework with the DIPP model and other parts of the framework remain the same. The results show that the GameFormer model still outperforms the DIPP model, as a result of better initial plans for the ego agent and prediction results for other agents.

Table S2. Comparison with DIPP planner on the nuPlan testing benchmark

Method	Overall	OL	CL non-reactive	CL reactivate
DIPP	0.7950	0.8141	0.7853	0.7857
Ours	**0.8288**	**0.8400**	**0.8087**	**0.8376**

B.3. Abalation Study

Effects of decoding levels on closed-loop planning. We investigate the influence of decoding levels on closed-loop planning performance in selected WOMD scenarios, using the success rate (without collision) as the main metric. We also report the inference time of the prediction network (without the refine motion planner) in closed-loop planning, which is executed on an NVIDIA RTX 3080 GPU. The results in Table S3 reveal that increasing the decoding layers could potentially lead to a higher success rate, and even adding a single layer of interaction modeling can bring significant improvement compared to level-0. In closed-loop

testing, the success rate reaches a plateau at a decoding level of 2, while the computation time continues to increase. Therefore, using two reasoning levels in our model may offer a favorable balance between performance and efficiency in practical applications.

Table S3. Effects of decoding levels on closed-loop planning

Level	Success rate (%)	Inference time (ms)
0	89.5	31.8
1	92.25	44.1
2	94	56.7
3	94.5	66.5
4	94.5	79.2

C. Additional Qualitative Results

C.1. Interaction Prediction

Fig. S1 presents additional qualitative results of our GameFormer framework in the interaction prediction task, showcasing the ability of our method to handle a variety of interaction pairs and complex urban driving scenarios.

C.2. Level-k Prediction

Fig. S2 illustrates the most-likely joint trajectories of the target agents at different interaction levels. The results demonstrate that our proposed model is capable of refining the prediction results in the iterated interaction process. At level-0, the predictions for target agents appear more independent, potentially leading to trajectory collisions. However, through iterative refinement, our model can generate consistent and human-like trajectories at a higher interaction level.

C.3. Open-loop Planning

Fig. S3 provides additional qualitative results of our model in the open-loop planning task, which show the ability of our model to jointly plan the trajectory of the AV and predict the behaviors of neighboring agents.

C.4. Closed-loop Planning

We visualize the closed-loop planning performance of our method through videos available on the project website, including interactive urban driving scenarios from both the WOMD and nuPlan datasets.

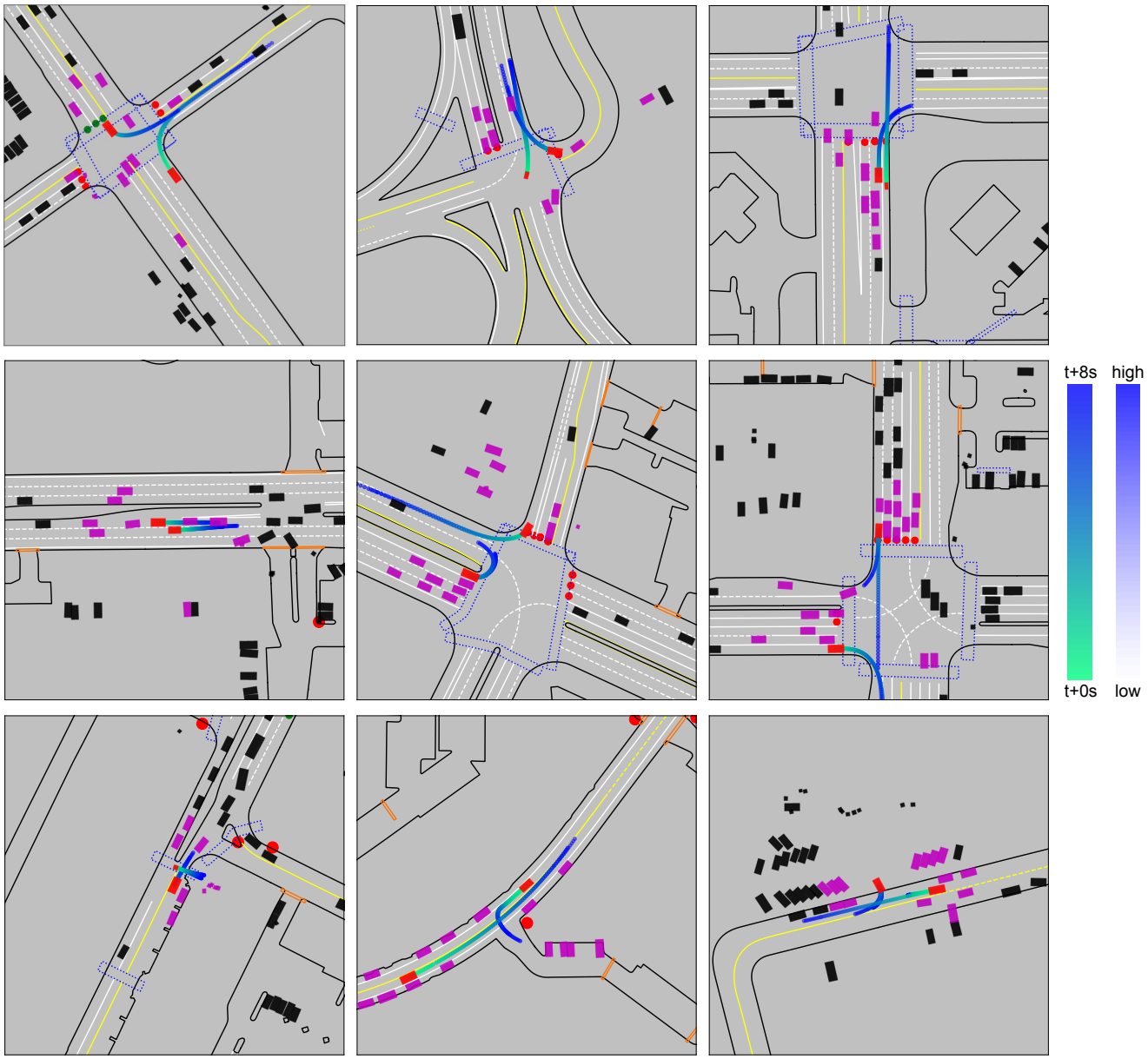

Figure S1. Additional qualitative results of interaction prediction. The red boxes are interacting agents to predict, and the magenta boxes are background neighboring agents. Six joint trajectories of the two interacting agents are predicted.

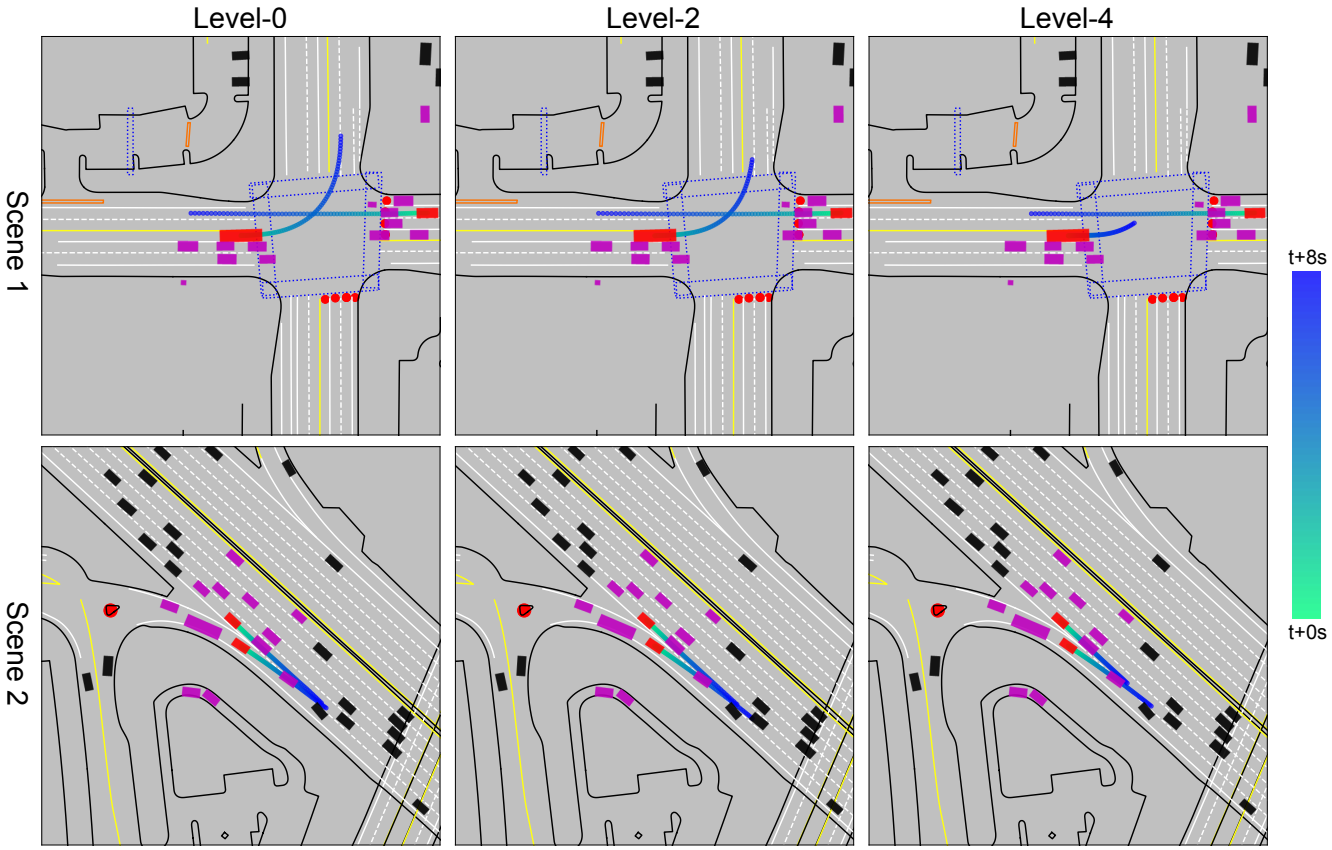

Figure S2. Prediction results of the two interacting agents at different reasoning levels. Only the most-likely joint trajectories of the target agents are displayed for clarity.

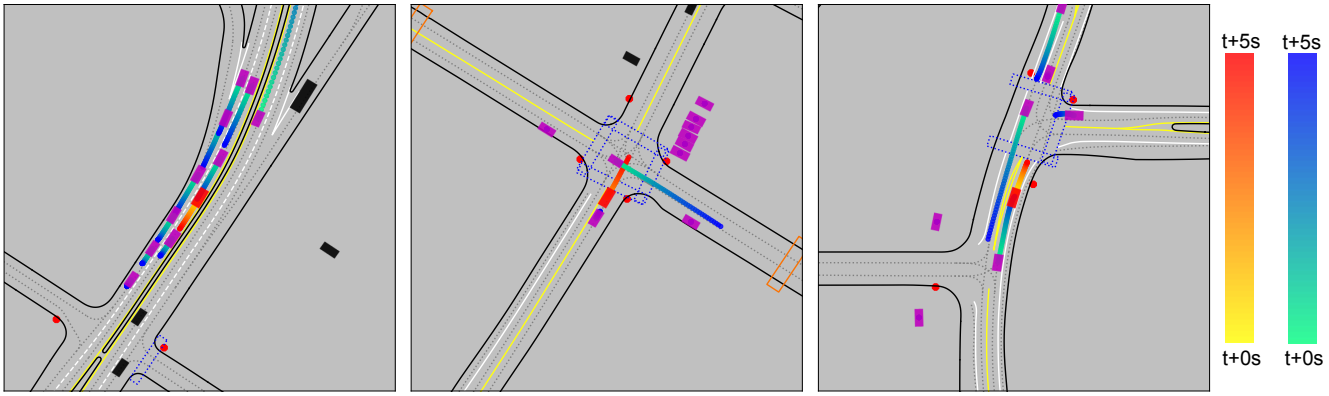

Figure S3. Additional qualitative results of open-loop planning. The red box is the AV and the magenta boxes are its neighboring agents; the red trajectory is the plan of the AV and the blue ones are the predictions of neighboring agents.